HIVE介绍

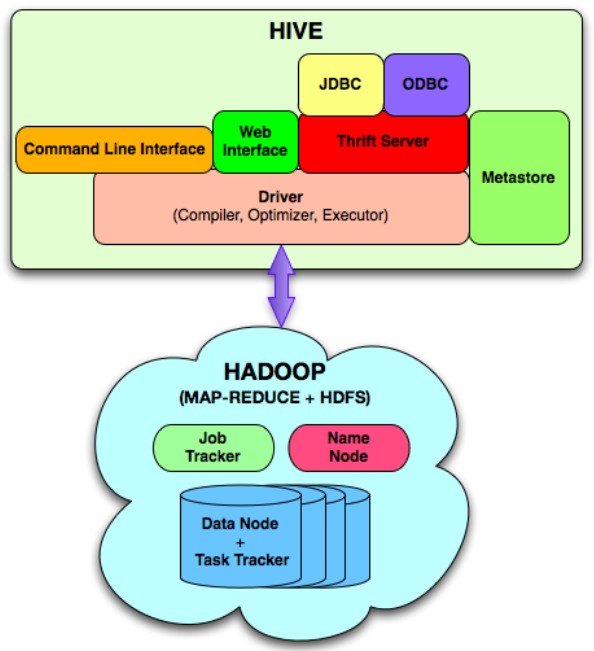
# 简介

## 是什么

hive是一个基于hadoop的数据仓库。使用hadoop-hdfs作为数据存储层；提供类似SQL的语言（HQL），通过hadoop-mapreduce完成数据计算；通过HQL语言提供使用者部分传统RDBMS一样的表格查询特性和分布式存储计算特性。

类似的系统有yahoo的pig[1] ，google的sawzall[2]，microsoft的DryadLINQ[3]。

## 架构



图表 1 hive架构图

1. 操作界面：CLI，Web，Thrift
2. driver：hive系统将用户操作转化为mapreduce计算的模块（重点）
3. hadoop：hdfs+mapreduce
4. metastore：存储元数据

## 语言

一般有DDL和DML两种：hive采用DDL方式和少量DML方式，类似sql；pig使用DML方式。

**DDL**：data definition language（只讲definition，不讲实现）

{create/alter/drop}{table/view/partition}

create table as select

**DML**：data manipulation language（有关于实现操作）

insert overwrite

**hive示例**

|  |  |
| --- | --- |
| 加载 | **load data** local input ‘/logs/urls.txt’ **into table** urls partition (ds=’2010-01-01’); |
| 写入 | **INSERT** **OVERWRITE** TABLE result |
| 操作 | **SELECT** category, AVG(pagerank)  **FROM** urls **WHERE** pagerank > 0.2  **GROUP BY** category**;** |

**pig示例**

|  |  |
| --- | --- |
| 加载 | urls = **LOAD** ‘/logs/urls.txt’ USING myLoad() AS (category,pagerank); |
| 操作 | good\_urls = **FILTER** urls BY pagerank > 0.2;  groups = **GROUP** good\_urls BY category;  output = **FOREACH** groups **GENERATE** category, AVG(good\_urls.pagerank); |
| 写入 | **STORE** output **INTO** ‘myoutput’ USING myStore(); |

**hive中使用自定义map-reduce**

|  |
| --- |
| FROM (  FROM pv\_users  **MAP** pv\_users.userid, pv\_users.date  **USING 'map\_script'**  AS dt, uid  CLUSTER BY dt) map\_output  INSERT OVERWRITE TABLE pv\_users\_reduced  **REDUCE** map\_output.dt, map\_output.uid  **USING 'reduce\_script'**  AS date, count; |

## 其他一些功能

1. 能够ALERT一个table，主要是add一个column。
2. 分区（partition）：
   1. 建表的时候指定分区方式：

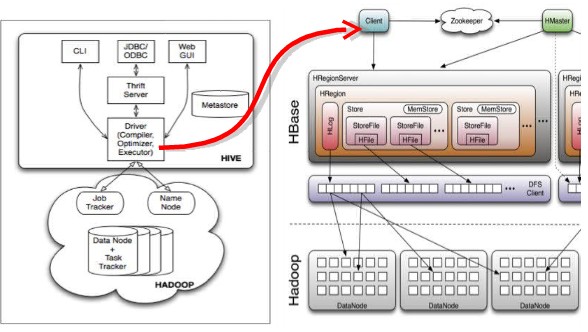
CREATE TABLE invites (foo INT, bar STRING) **PARTITIONED** BY (**ds** STRING);

* 1. 导入的时候指定分区依据：

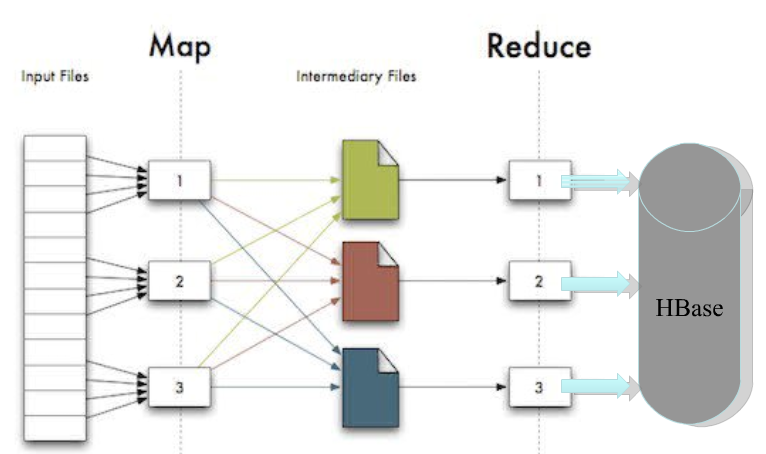
LOAD DATA LOCAL INPATH './examples/files/kv2.txt' OVERWRITE INTO TABLE invites **PARTITION** (**ds**='2008-08-15');

LOAD DATA LOCAL INPATH './examples/files/kv3.txt' OVERWRITE INTO TABLE invites **PARTITION** (**ds**='2008-08-08');

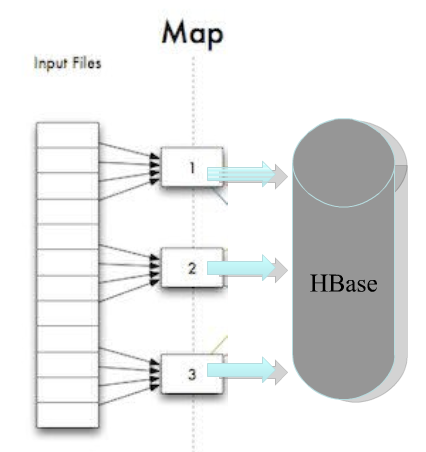
1. 类似 select \* from tbl 的查询不需要 MapReduce。
2. hive不只是可以mapreduce



图表 2 hive结合HBase的逻辑图[5]”



图表 3 reduce阶段写入HBase的方式[5]”



图表 4 map-only job写入HBase的方式[5]”

# 实现

## 原数据(Metadata)

hive的元数据存储在传统的RDBMS中，现在为mysql中。采用JDO（JPOX）。

**原因**：访问这些Metadata，我们想要“很低的延时”，而存在hdfs中是无法满足。（元数据对hive是相当重要的，因此一般要求有备份机制）

**使用**：元数据都是在HQL语句编译的时候，就被生成一个xml文件（包含此次编译所有需要的元数据信息）存储在hdfs中，然后运行mapreduce时传递给mapper和reducer。（减少后期访问）

## 查询解析(query parser)

这一步是实现中最主要的操作，即架构图中Driver的大部分。下面将具体介绍其中的每一个小步。

### 解析(parse)

使用antlr解析HQL语句，并产生AST（abstract syntax tree）。

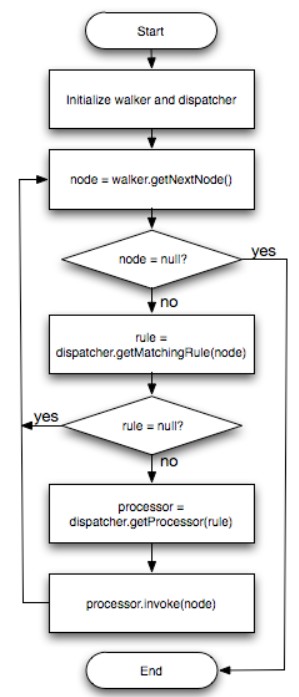
### 类型检测和语义分析

分析所有输入输出的table，并创建logical-plan。通过一种中间表示结构query block（QB）tree，将AST转换成operator-DAG：将嵌套的queries变成父子关系的QB-tree。

### 优化(Optimization)

通过operator-DAG的中“元素的前后满足关系”生成一些操作（operator）。主要的五个元素为：Node, GrahpWalder, Dispatcher, Rule, Processor：

**GraphWalker**遍历（walk）DAG中所有的**Node**，并检查一个**Rule**是否满足，在满足的条件下回出发一个对应的**Processor**。**Dispatcher**则维护Rule到Processor的映射，并进行Rule的匹配工作。



图表 5 优化过程中的典型转换流图[4]

**简单的几个优化步骤**

针对优化，这里给出了一些简单的处理方式：

1. 列裁剪（**Column pruning**）：只有需要用到的列才进行输出
2. 谓词下推（**Predicate pushdown**）：尽早进行数据过滤(见图表 6中，下面为先处理的逻辑)，减少后续处理的数据量
3. 分区裁剪（**Partition pruning**）：只读取满足分区条件的文件
4. **map-join**：对于join中一些小文件，可以在map阶段进行join操作，见3.2.2节map-join部分
5. **join-reordering**：将在reducer中进行join操作时的小table放入内存，而大table通过stream方式读取
6. **Group-by**优化： 进行局部聚合进行优化（包括hash-based和sort-based），对于skew的key（key的row num和size在reduce时非常不均）可以进行两次map-reduce的方式优化

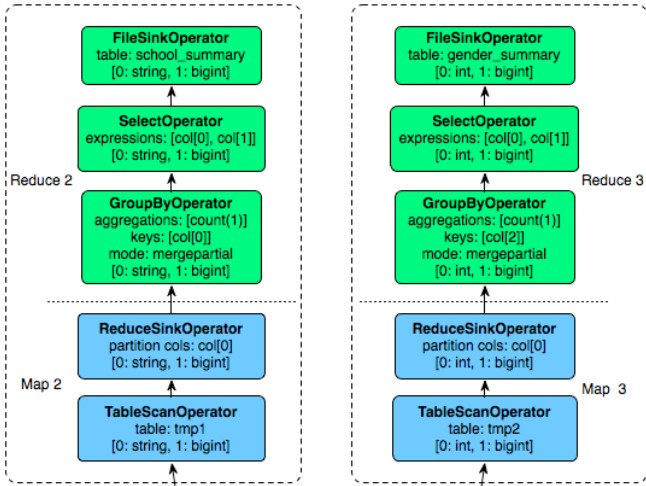
**说明**：基本上用于优化的提示（hint）都是一些配置项，map-join除外，需要具体在HQL直接指定。

### physical plan的生成

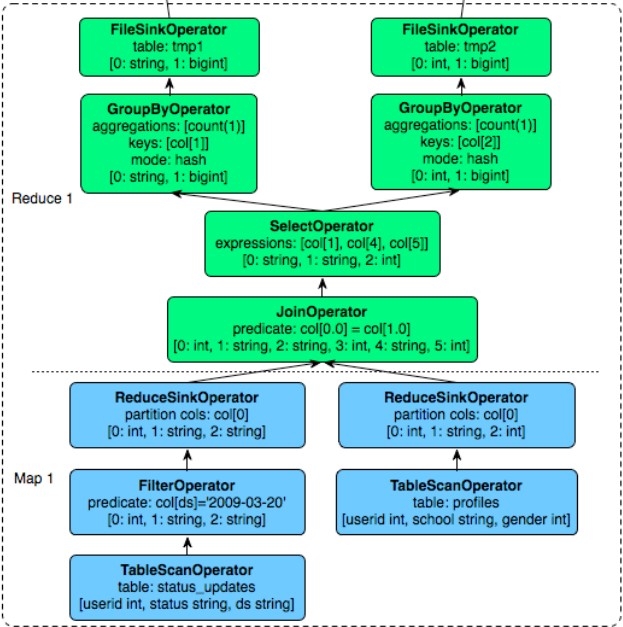
根据上一步的结果，分解成一些map/reduce操作，并将最终结果（即一些plan的xml文件）写入到hdfs。

这里给出一个论文[4]中的例子：

|  |
| --- |
| **FROM** (SELECT a.status, b.school, b.gender  FROM status\_updates a JOIN profiles b  ON (a.userid = b.userid AND a.ds='2009-03-20' )) subq1  **INSERT** OVERWRITE TABLE gender\_summary PARTITION(ds='2009-03-20')  **SELECT** subq1.gender, COUNT(1) GROUP BY subq1.gender  **INSERT** OVERWRITE TABLE school\_summary PARTITION(ds='2009-03-20')  **SELECT** subq1.school, COUNT(1) GROUP BY subq1.school |



图表 6 有3个job的多表插入查询的query-plan(1)



图表 7 有3个job的多表插入查询的query-plan(2)

**简单说明**

map1+reduce1将生成的数据分别写入两个临时的hdfs文件tmp1和tmp2，map2+reduce2和map3+reduce3就需要等待tmp1和tmp2的输出才能运行。

**一些理解和疑问**

1. 为什么map-reduce1中会放入GroupByOperator和FileSinkOperator？

A ：是predicate pushdown的结果

1. sink不知道什么意思？

A ：我理解成map/reduce中emit函数的操作

1. 中间selectOperator和JoinOperator操作分成了两步

A ：应该为了逻辑上的分开处理

**hive计划**

Hive使用了rule-based的优化方案，简单但不够优秀。后期计划是建立cost-based的优化方案。

## 执行引擎(Execution Engine)

根据job间的依赖的顺序执行任务。一个mapreduce-job首先是被编写成一个plan.xml文件，运行时先解析plan.xml，然后用hadoop运行。

# 其他说明及优化

## 数据模型

Hive 中包含以下**数据模型**：Table，External Table，Partition，Bucket。

1. Hive 中的 **Table** 和数据库中的 Table 在概念上是类似的，每一个 Table 在 Hive 中都有一个相应的目录存储数据。例如，一个表 pvs，它在 HDFS 中的路径为：/wh/pvs，其中，wh 是在 hive-site.xml 中由 ${hive.metastore.warehouse.dir} 指定的数据仓库的目录，所有的 Table 数据（不包括 External Table）都保存在这个目录中。
2. **External Table** 指向已经在 HDFS 中存在的数据，可以创建 Partition。它和 Table 在元数据的组织上是相同的，而实际数据的存储则有较大的差异。
   1. **Table** 的创建过程和数据加载过程（这两个过程可以在同一个语句中完成），在加载数据的过程中，实际数据会被移动到数据仓库目录中；之后对数据对访问将会直接在数据仓库目录中完成。删除表时，表中的数据和元数据将会被同时删除。
   2. **External Table** 只有一个过程，加载数据和创建表同时完成（CREATE EXTERNAL TABLE ……LOCATION），实际数据是存储在 LOCATION 后面指定的 HDFS 路径中，并不会移动到数据仓库目录中。当删除一个 External Table 时，仅删除元数据。
3. **Partition** 对应于数据库中的 Partition 列的密集索引，但是 Hive 中 Partition 的组织方式和数据库中的很不相同。在 Hive 中，表中的一个 Partition 对应于表下的一个目录，所有的 Partition 的数据都存储在对应的目录中。例如：pvs 表中包含 ds 和 city 两个 Partition，则对应于 ds = 20090801, ctry = US 的 HDFS 子目录为：/wh/pvs/ds=20090801/ctry=US；对应于 ds = 20090801, ctry = CA 的 HDFS 子目录为；/wh/pvs/ds=20090801/ctry=CA。[PARTITIONED BY]
4. **Buckets** 对指定列计算 hash，根据 hash 值切分数据，目的是为了并行，每一个 Bucket 对应一个文件。如将 user 列分散至 32 个 bucket，首先对 user 列的值计算 hash，对应 hash 值为 0 的 HDFS 目录为：/wh/pvs/ds=20090801/ctry=US/part-00000；hash 值为 20 的 HDFS 目录为：/wh/pvs/ds=20090801/ctry=US/part-00020。[CLUSTERED BY]

关于HQL语言使用以及其他hive内容，见[3]

## 功能及优化

### PARTITION

**功能**

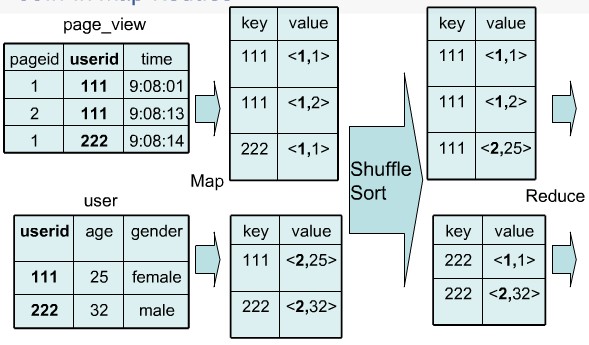
除了在创建table时指定partition，用户可以用 ALTER TABLE ADD PARTITION 来向一个表中增加分区。当分区名是字符串时加引号。也可以用 ALTER TABLE DROP PARTITION 来删除分区。分区的元数据和数据将被一并删除。

**借鉴**

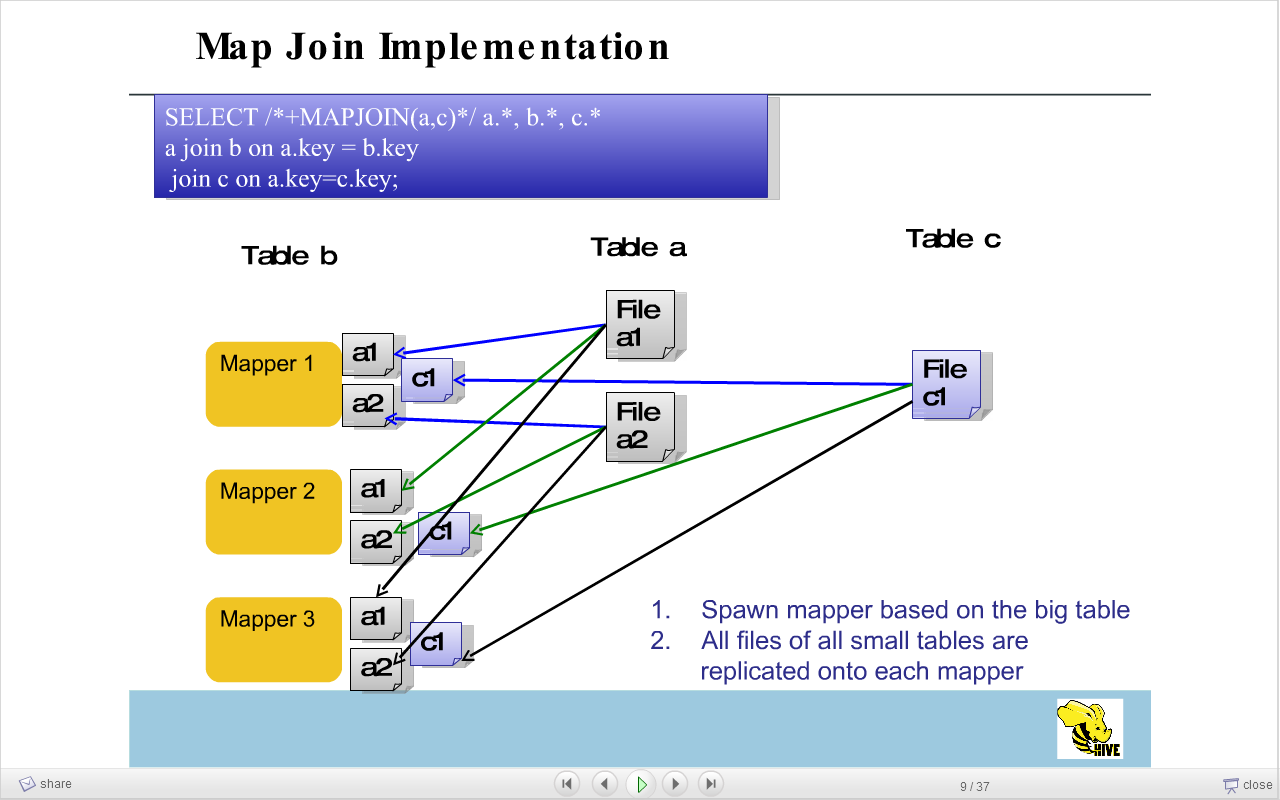
hive使用的是Range-partition，也可以参照MYSQL的LIST-partition，也就是将partition表达式”ds=2010-01-01”的**等式表达式**变成**任意的函数表达式**in\_list(list\_id)，参考[6]”[7]” 。

### JOIN

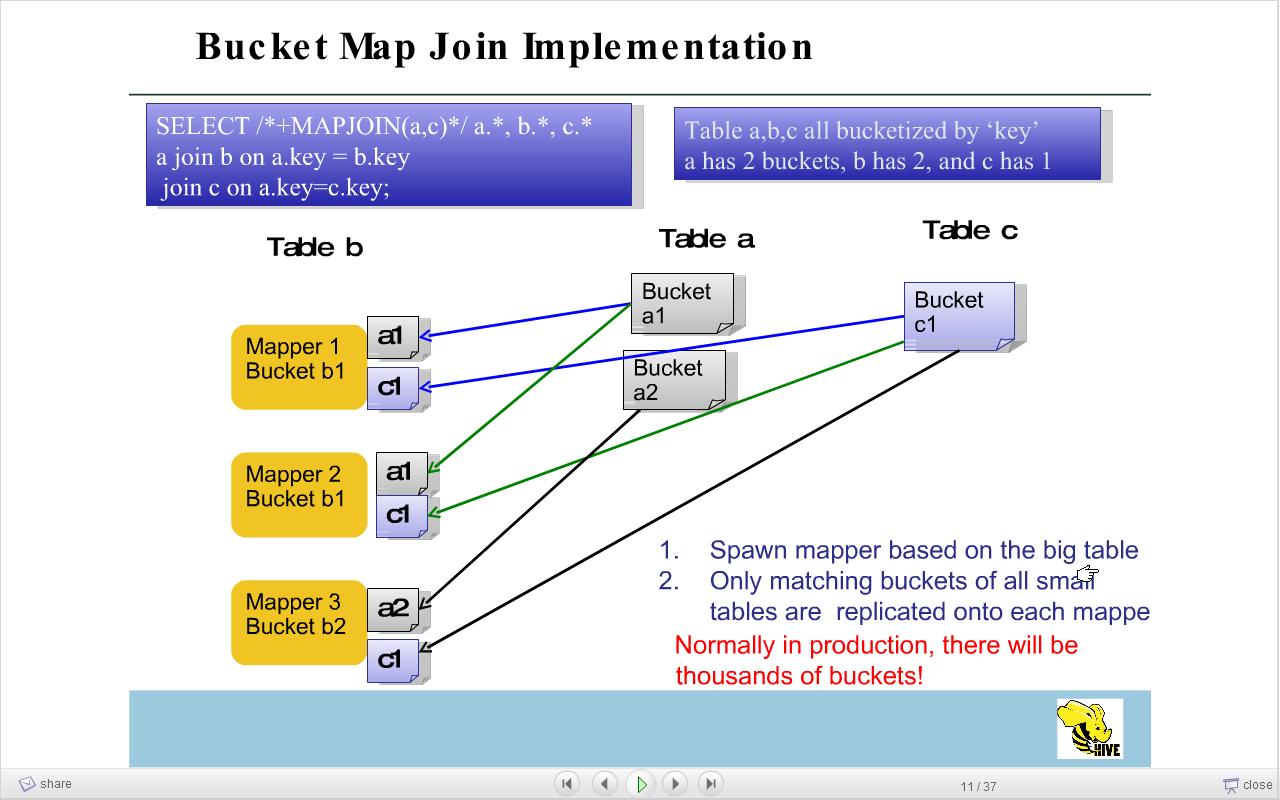
1. Hive 只支持**等值连接**（equality joins）、**外连接**（outer joins）和**left semi joins**。**Hive 不支持所有非等值的连接**，因为非等值连接非常难转化到 map/reduce 任务。join的实现见图表 8，另外Hive 支持多于 2 个表的连接。
   * join LEFT/RIGHT OUTER ：一定输出左边/右边的每一行对应的结果
   * left semi join用于实现a.key in select key from table b(即in/exist功能)
2. 多个表的join：
   * 多表join 时，每次 map/reduce 任务的逻辑是这样的：reducer 会缓存 join 序列中除了最后一个表的所有表的记录，再通过最后一个表将结果序列化到文件系统。这一实现有助于在 reduce端减少内存的使用量。实践中，应该把最大的那个表写在最后（否则会因为缓存浪费大量内存）。
   * 多个表的 join key 是同一个时， join 会被转化为**单个** map/reduce 任务：Reduce 端会缓存 a 表和 b 表的记录，然后每次取得一个 c 表的记录就计算一次 join 结果
   * 不同join key时，会被转化为**多个** map/reduce 任务：第一次缓存 a 表，用 b 表序列化；第二次缓存第一次 map/reduce 任务的结果，然后用 c 表序列化
3. MAP-JOIN：允许在map阶段进行join操作，即需要在map时先加载一个join的数据到内存，然后直接过滤这片buff，输出结果。
   * 实现
     1. SELECT **/\*+ MAPJOIN(t2) \*/** t1.c1, t2.c1 FROM t1 JOIN t2 ON(t1.c2 = t2.c2);
     2. 通过打tag来实现同key的join（如果多个join都一样的key，就打多个tag）
     3. 小文件被复制多分到每个split后，优化map-join（指定size和num rows），见图表 9
   * 优化
     1. **bucket-join**：在特殊情况下，对大文件a，小文件b进行bucket分桶（b文件不是很小的时候），减少每个a-split上都需要一个b（现在只需要一个b-bucket就可以了，但需要按key排序分桶的），见图表 10
     2. **sort-join**: 当文件都比较大，可以边读边扔（因为是排序的，有点像merge sort），就可以处理大文件了。
     3. **hash-join**:在map端设计一个hash，当达到一定大小（比如50%hash满），进行一次计算输出
4. 其他join
   * skew-join：针对不清楚A还是B的某个key的size小（或者交替），可以的方法有：将A join B时A的key的size大的保存输出到另一个结果（生成第二个mapreduce来将B的key放buf），见图表 12



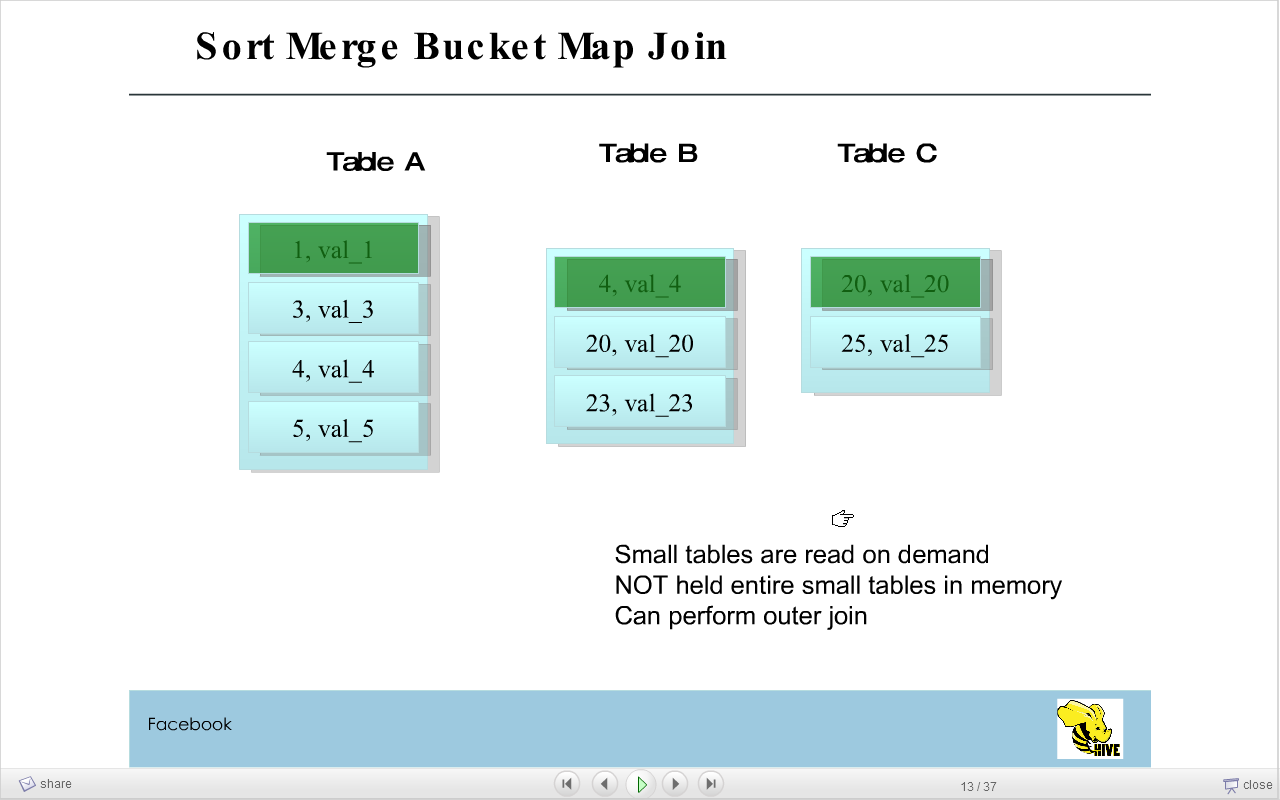
图表 8 join实现[8]



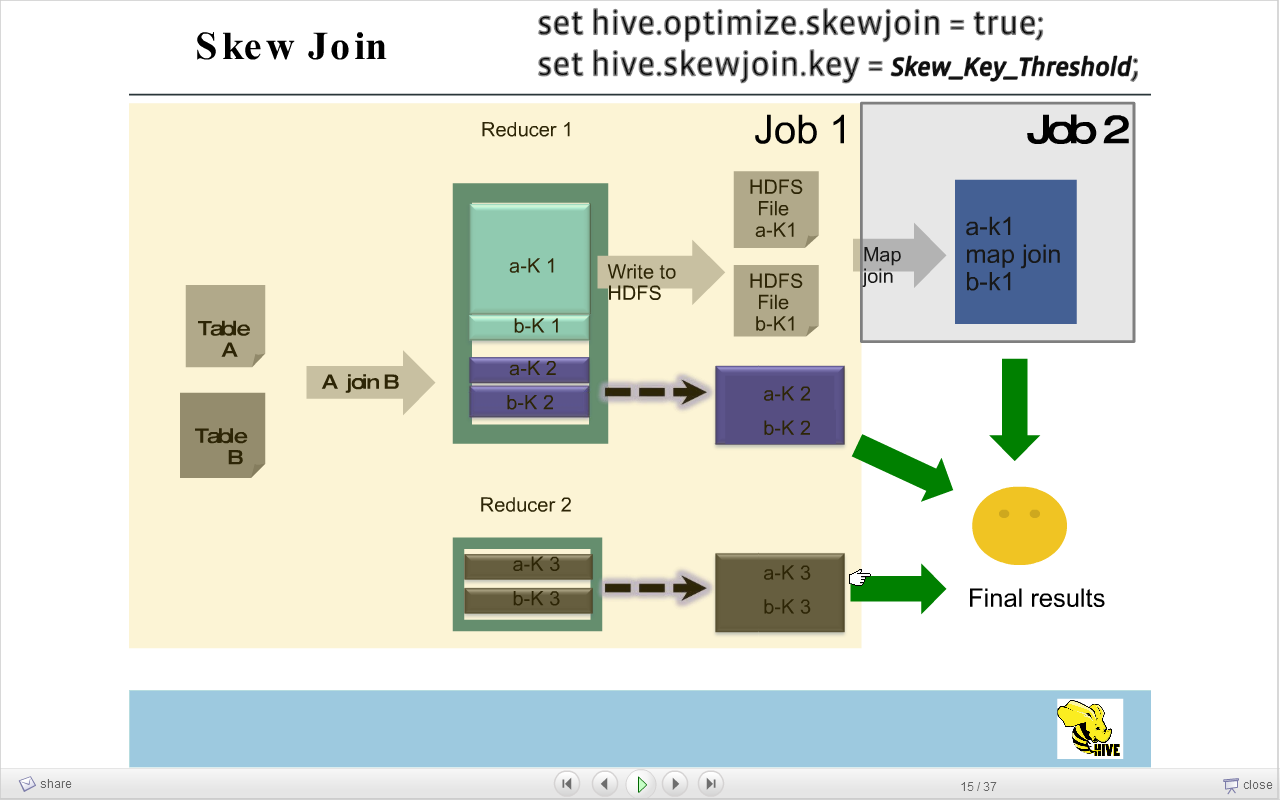
图表 9 普通map-join实现[9]



图表 10 bucket map-join[9]



图表 11 sort-based map-join[9]

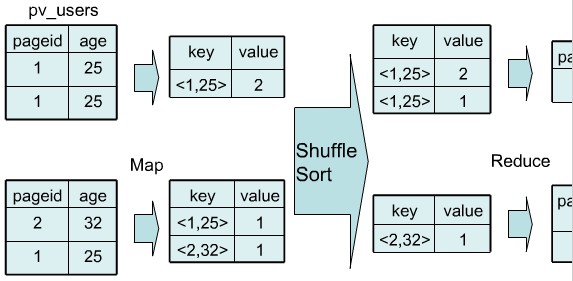


图表 12 skew map-join[9]

### GROUP BY

group-by的实现过程见图表 13。

1. **局部聚合**：
   1. sort-based(combiner)：一般的group by都可以先在map端做combiner操作(如count等函数)
   2. Hash-based：在map阶段通过保存hash来进行早期的聚合操作（类似combine，但粒度更小）。配置变量为：hive.map.aggr.hash.percentmemory
2. **两次job（负载均衡）**：为了减少一个因key分布不均导致某些key数据太多，可以要求生成两个MR-job：第一阶段，随机分布key（或者选取其他列的分桶方式），进行一次Group-by后；第二次进行reducer时，单个key的rownum一般就不会很大了。两个配置变量为：hive.mapjoin.size.key，hive.mapjoin.cache.numrows



图表 13 group-by实现[8]

### SORT BY

由于reduce是按照key分桶排序，当sort by为多列时，可能会出现相同第一个列的会在不同reducer中（因为多个列总的作为一个key）。解决方法是使用DISTRIBUTE BY，指定需要划分在一个reducer中的那些列（或叫做子key）。也就是partitioner-key和sort-key不相同。

## 存储格式

hive允许多种on-disk格式(可自定义)：

1. file format：row-based，column-based，block-based
2. raw format：text-based，binary-based，custom-based
3. index format？

多种in-memory格式(可自定义)：

Integer，LazyInteger，String，Text

# 参考

[1] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, “Pig latin,” *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD ’08*, 2008, p. 1099.

[2] R. Pike, S. Dorward, and R. Griesemer, Quinlan, “Interpreting the data: Parallel analysis with sawzall,” *Scientific Programming Journal, Special Issue on Grids and Worldwide Computing Programming Models and In*, vol. frastructu, p. //labs.

[3] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, “DryadLINQ: A System for General-Purpose Distributed Data-parallel Computing Using a High-Level Language,” *ACM SIGOPS Operating Systems Review*, vol. 41, Jun. 2009, p. 59.

[4] A. Thusoo, J.S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, and H. Liu, “Hive – A Petabyte Scale Data Warehouse Using Hadoop,” *Architecture*.

[5] J. Sichi, Facebook Hive team, “Hive/HBase Integration.”

[6] “MYSQL-reference manual.”

[7] “Partition(database).”

[8] N.Z. Ashish Thusoo, Raghotham Murthy , Joydeep Sen Sarma, Zheng Shao, Namit Jain, Prasad Chakka, Suresh Anthony, Hao Liu, “Hive - A Petabyte Scale Data Warehouse Using Hadoop,” 2010.

[9] F.H. Team, “Hive New Features and API.”

**web page**(对应前面的paper名字的网页)

Hive/HBase Integration

<http://www.slideshare.net/hadoopusergroup/hive-h-basehadoopapr2010>

Hive - A Petabyte Scale Data Warehouse Using Hadoop

<http://www.slideshare.net/ragho/hive-icde-2010>

Hive New Features and API

<http://www.slideshare.net/zshao/hive-user-meeting-march-2010-hive-team>

MYSQL-reference manual

<http://dev.mysql.com/doc/refman/5.1/en/partitioning-list.html>

Partition(database)

<http://en.wikipedia.org/wiki/Partition_(database)>

**附加说明**

一些相关的ppt和paper在：http://wiki.apache.org/hadoop/Hive/Presentations